REMARKS

Claims 1-9 are pending. Claim 1 was amended to more particularly point out and distinctly claim the present invention. Support for the amended language is provided in at least Figure 11 and page 23, line 12 through page 24, line 19 of the present application. Accordingly, no new matter has been entered.

Claims 1 and 6 were amended to improve their form and to correct minor grammatical errors. Claims 1 and 6 were further amended to remove certain limitations that are not believed to be necessary for patentability, and such limitations were moved to new dependent claims 7-9. The moved limitations were also modified slightly to improve their form.

Withdrawal of all rejections of the pending claims is respectfully requested for at least the reasons set forth below.

Request for Interview Prior to Formal Action on Amendment

Applicants request an interview prior to formal action on this amendment. An "Applicant Initiated Interview Request Form" accompanies this paper. Please contact Applicants' undersigned representative to schedule the interview.

Rejection under 35 U.S.C. § 102(e)

Claims 1-6 were rejected under 35 U.S.C. § 102(e) as allegedly being anticipated by Subramanian. Applicants respectfully request withdrawal of this rejection as it pertains to the amended set of claims.

1. Present invention

Amended claim 1 of the present invention recites, in part, an upgradeable and extendable wireless communication system having a plurality of layers, wherein each layer includes a plurality of configurable computational units that implement operation of wireless digital communication functions. At least one of the plurality of configurable computational units is dynamically selected in real-time based on a wireless communication standard to configure various hardware for dedicated functions.

One preferred embodiment of amended claim 1 is described, in part, in Figure 11 and on page 23, line 12 through page 24, line 19 of the present specification, which reads as follows (underlining added for emphasis):

> If the available resources maintained by hardware manager 632 are sufficient to implement the new algorithm profile, hardware manager 632 starts the operation of kernel synthesis (1126). Kernel synthesis is a real-time procedure in which optimized hardware macros and software tasks are chosen to implement the desired algorithm. There are two libraries to provide the required optimized hardware macros and software tasks. One is a macro-based library (1128) which supports different classes of optimized hardware macros provided by different configurable devices, available from CPLD vendors such as ALTERA, Xilinx, and Lucent. The other library is a subroutine based library (1130) which supports many different classes of optimized software subroutines provided by different programmable devices, available from DSP vendors like such as TI, Motorola, and ADI. After choosing suitable optimized macros from macro-based library (1128) and software tasks from subroutine based library (1130), the new algorithm can be synthesized. Configure driver 826 of hardware manager 632 now performs mapping to the supporting driver (1132). Moreover, download driver 828 of hardware manager 632 is invoked to reconfigure and reprogram the corresponding targeted configurable kernels and programmable kernels (1134).
>
> This execution procedure of dynamic hardware configuration 1100 is a closed-loop procedure including performance measuring, new algorithm profile generation, radio resource utilization checking, kernel synthesis for new algorithm profile, and targeted kernels reconfiguring and reprogramming. Via the cooperation of radio resource utilization control driver 822, configure driver 826, and download driver 828, hardware resources can be utilized more efficiently. Further, the capability of the performance feedback-decision resource management mechanism can be achieved by implementing this closed-loop execution procedure of dynamic hardware configuration and finally download executable files to configurable devices with local reconfiguring ability. Using the performance feedback-decision resource management mechanism, the current status of the radio link can be sensed immediately and radio link performance related parameters like SNR and CIR are fed back via performance counter 726 to decide whether the utilization of resources should be

> <u>changed dynamically to satisfy the variable characteristic of the radio link and thereby approach the goal of incremental optimization.</u>

Amended claim 6 of the present invention recites a method of programming and configuring components of an upgradeable and extendable wireless communication system in order to implement multiple wireless communication standards, services, and applications. The method comprises, in part, (a) identifying one of the application, standard or service to be implemented, and (b) <u>compiling[1] software</u> stored in a software library that is associated with the identified application, standard or service and storing the compiled software in a host memory. See, Appendix A for an illustration that accompanies a definition of "compile."

One preferred embodiment of amended claim 6 is described, in part, in Figure 8 and on page 17, line 12 through page 18, line 3 of the present specification, which reads as follows (underlining added for emphasis):

> FIG. 8 illustrates an execution procedure 800 practiced in a hardware configuration for an application program. A dynamic library 802 includes radio function signal processing modules, real-time control modules and hardware interface modules. Library 802 has linking information to the real functions at run time. This technology is similar to a dynamic linking library which has been used in operating systems. <u>An application program 804, such as a wideband code-division multiple access (WCDMA) or wireless local area network (WLAN), can be compiled into an executable file 806 by a compiler 808.</u> The loading of dynamic library 802 entails a reconfiguration manager 810 downloading a new library from a network or loading the library from a storage device to hardware devices. Required resource parameters 812 generated by executable file 806 are sent to reconfiguration manager 810 to generate an application API profile 814 for

[1] "compile" definition from http://www.webopedia.com, downloaded on June 8, 2005: To transform a program written in a high-level programming language from source code into object code. Programmers write programs in a form called source code. Source code must go through several steps before it becomes an executable program. The first step is to pass the source code through a compiler, which translates the high-level language instructions into object code.

The final step in producing an executable program -- after the compiler has produced object code -- is to pass the object code through a linker. The linker combines modules and gives real values to all symbolic addresses, thereby producing machine code.

that specific application. Meanwhile, executable file 806 at run time, including linking information to hardware-related programs such as DSP codes and HDL codes, of a specific air-interface application, is generated from the compiled application program and the radio functions linked from library 802.

2. Subramanian

Subramanian discloses a digital wireless communication device that includes:

(a) a software-programmable processor,

(b) a heterogeneous reconfigurable multiprocessing logic circuit, and

(c) a bus connecting the software-programmable processor and the heterogeneous reconfigurable multiprocessing logic circuit.

The software-programmable processor is selected from the group comprising a digital signal processor and a central processing unit. The heterogeneous reconfigurable multiprocessor comprises a set of heterogeneous signal processing kernels and a reconfigurable data router interconnecting the heterogeneous signal processing kernels. The signal processing kernels and data router are controlled by the software-programmable processor via control busses.

Although Subramanian discloses interconnecting signal processing kernels (which the Examiner argues are equivalent to the claimed "plurality of configurable computation units"), Subramanian does not disclose or suggest that the signal processing kernels are dynamically selected in real-time based on a wireless communication standard. Applicant has carefully read all of the text portions and figures highlighted by the Examiner and cannot find any text portions or figures that disclose or suggest that Subramanian's signal processing kernels are dynamically selected in real-time.

Subramanian discloses that configuration or reconfiguration can occur in the factory or field through various means such as factory/point-of-sale programming, remote control, and over-the-air or over-the-network download (column 3, lines 42-45 and column 10, lines 2-5). However, this feature does not disclose or suggest dynamic selection of the signal processing kernels in real-time. Instead, this feature merely describes where configuration/reconfiguration can occur and what signal paths can be used.

Furthermore, Subramanian's software modules are in the form of <u>object code</u>[2]. See, Appendix B for a discussion of object code which clearly illustrates and describes that object code is compiled software source code. That is, object code is code produced by a compiler, and thus <u>no compiler is needed in Subramanian</u>. In fact, no compiler is disclosed or suggested in Subramanian.

3. <u>Examiner's Office Action</u>

Claim 1: In the outstanding Office Action, the Examiner equates the set of heterogeneous signal processing kernels to the claimed configurable computational units and asserts that Subramanian discloses computational units that are <u>dynamically selected</u> based on a wireless communication standard. The Examiner cites numerous text portions and figures in Subramanian that allegedly disclose this feature. As discussed above, none of these text portions or figures disclose or address <u>dynamic, real-time selection</u>.

Claim 6: In the outstanding Office Action, the Examiner refers to numerous text portions of Subramanian that allegedly disclose compiling software. Applicants have carefully reviewed the text portions and cannot find any disclosure or suggestion of a <u>compiling step</u> in Subramanian. In fact, Subramanian only uses object code, executable code, and executive code (which is described as being a segment of the microprocessor executable programs), all of which are compiled code (i.e., code that has already been compiled).

[2] "object code" definition from http://www.webopedia.com, downloaded on June 8, 2005: The code produced by a compiler. Programmers write programs in a form called source code. The source code consists of instructions in a particular language, like C or FORTRAN. Computers, however, can only execute instructions written in a low-level language called machine language.

To get from source code to machine language, the programs must be transformed by a compiler. The compiler produces an intermediary form called object code. Object code is often the same as or similar to a computer's machine language. The final step in producing an executable program is to transform the object code into machine language, if it is not already in this form. This can be done by a number of different types of programs, called assemblers, binders, linkers, and loaders.

In sum, Subramanian completely lacks at least the above-identified limitations in claims 1 and 6.

4. Patentability of amended claims 1 and 6 over Subramanian

Amended claims 1 and 6 each recite at least the following limitations that are not disclosed in Subramanian (underlining added for emphasis):

> (a) a plurality of configurable computational units that implement operation of wireless digital communication functions, at least one of the plurality of configurable computational units being dynamically selected in real-time based on a wireless communication standard to configure various hardware for dedicated functions (claim 1)
>
> (b) compiling software stored in a software library that is associated with the identified application, standard or service and storing the compiled software in a host memory (claim 6)

Accordingly, claims 1 and 6 cannot be anticipated by Subramanian. Nor is either of these limitations obvious in view of Subramanian. It would require a substantial reconstruction of Subramanian to incorporate real-time dynamic selection of Subramanian's signal processing kernels and a software compiling process.

5. Patentability of dependent claims

The dependent claims are believed to be patentable because they depend from allowable independent claims and because they recite additional patentable features.

Entry of Rule 116 Response

Entry of this response is requested because such response does not raise any new issues that would require further consideration and/or search. No new matter is raised by this response. This response could not have been previously presented because the outstanding rejection provides new explanations for the grounds of rejection. Lastly, it is requested that the response be entered even if the application is not allowed because this response will place the application in better form for appeal by materially simplifying the issues.

If the application is not in proper form for allowance, Applicants request that the Examiner telephone the undersigned to discuss any further outstanding issues.

Conclusion

Insofar as the Examiner's rejections were fully addressed, the instant application is in condition for allowance. Entry of this Rule 116 Response and issuance of a Notice of Allowability of all pending claims is therefore earnestly solicited.

Respectfully submitted,

PANGAN TING et al.

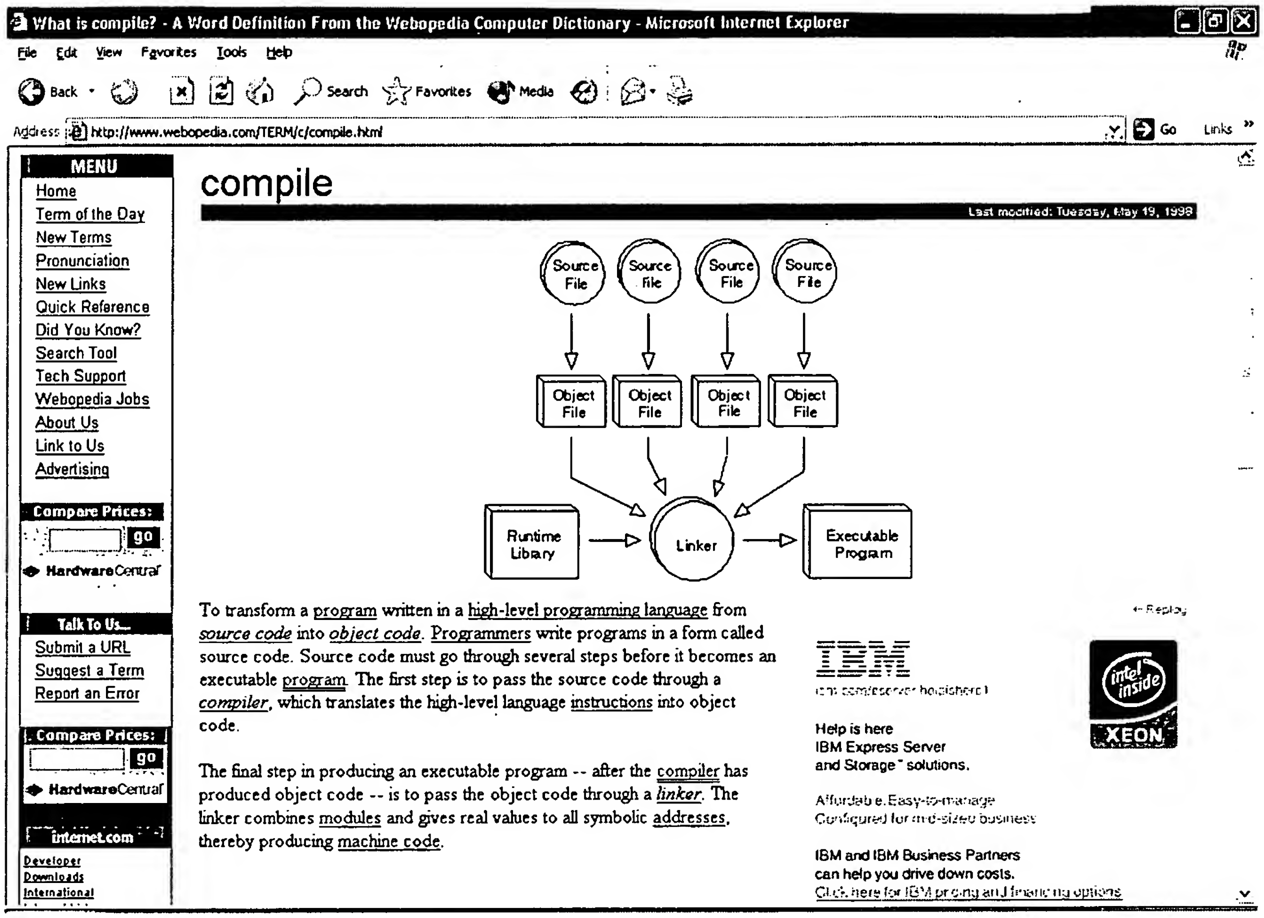
June 13, 2005 By: Clark Jablon
(Date)

CLARK A. JABLON
Registration No. 35,039
AKIN GUMP STRAUSS HAUER & FELD LLP
One Commerce Square
2005 Market Street - Suite 2200
Philadelphia, PA 19103
Direct Dial: (215) 965-1293
Facsimile: (215) 965-1210

7417035 v1

APPENDIX A

(Attachment to Reply to Office Action dated February 15, 2005
for U.S. Application No. 10/073,933)



compile

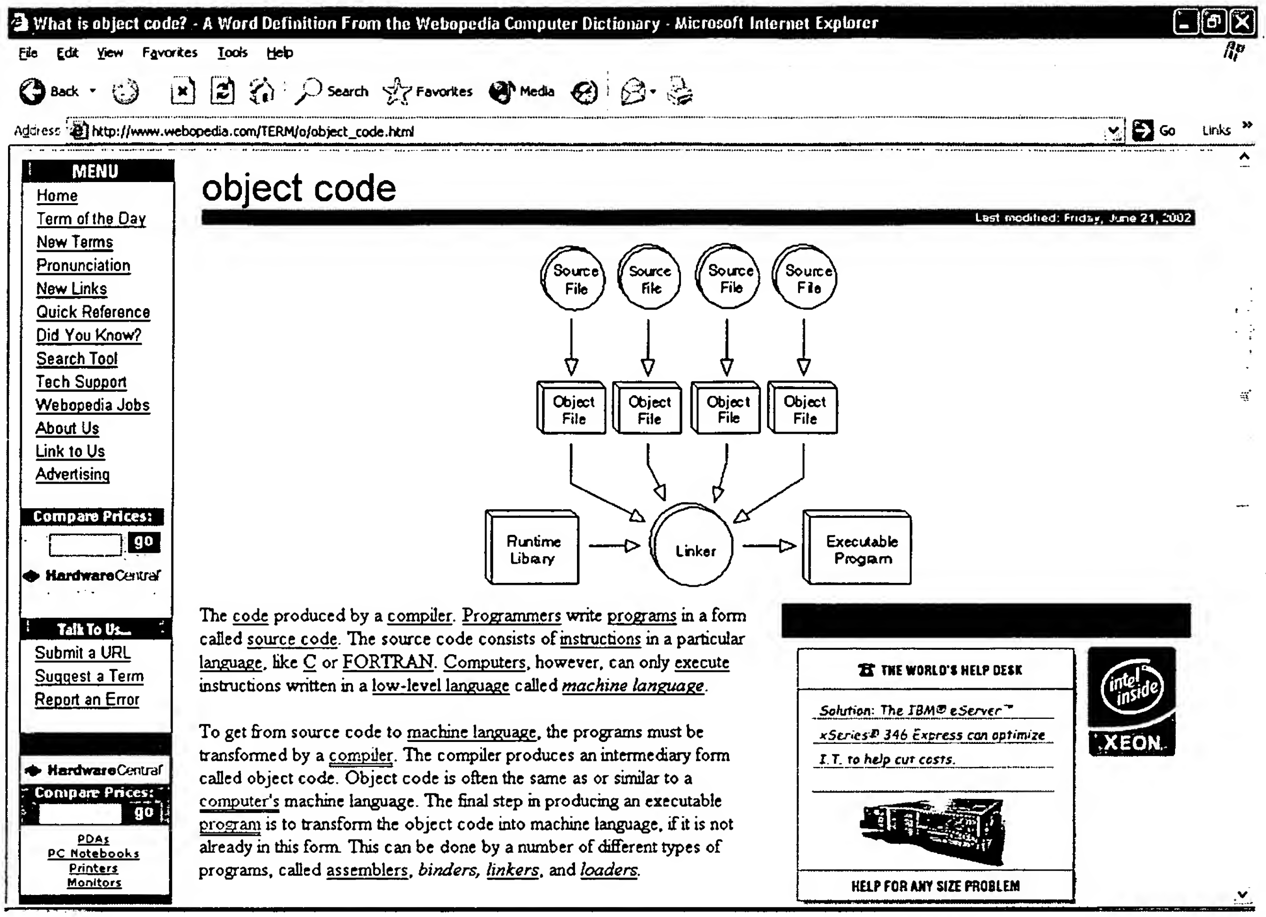
Last modified: Tuesday, May 19, 1998

To transform a program written in a high-level programming language from *source code* into *object code*. Programmers write programs in a form called source code. Source code must go through several steps before it becomes an executable program. The first step is to pass the source code through a *compiler*, which translates the high-level language instructions into object code.

The final step in producing an executable program -- after the compiler has produced object code -- is to pass the object code through a *linker*. The linker combines modules and gives real values to all symbolic addresses, thereby producing machine code.

APPENDIX B

(Attachment to Reply to Office Action dated February 15, 2005
for U.S. Application No. 10/073,933)



object code

Last modified: Friday, June 21, 2002

The code produced by a compiler. Programmers write programs in a form called source code. The source code consists of instructions in a particular language, like C or FORTRAN. Computers, however, can only execute instructions written in a low-level language called *machine language*.

To get from source code to machine language, the programs must be transformed by a compiler. The compiler produces an intermediary form called object code. Object code is often the same as or similar to a computer's machine language. The final step in producing an executable program is to transform the object code into machine language, if it is not already in this form. This can be done by a number of different types of programs, called assemblers, *binders*, *linkers*, and *loaders*.